CoLink: A Programming Framework for Decentralized Data Science
Final Report (Group Name: CoLink Dev)

Xiaoyuan Liu
xiaoyuanliu@berkeley.edu

Tianchen Liu
tianchenliu@berkeley.edu

Bryce Wong
brywong@berkeley.edu

Siyuan Zhuang
source@berkeley.edu

Abstract

Decentralized data collaboration is a common need in many research fields but it faces many deployment challenges, including practicability, compatibility and publicity. In this work, we propose CoLink to simplify the deployment of decentralized data science solutions. Extending gRPC, we allow cryptographic implementations in different programming languages to work together consistently. The evaluation case study shows the conciseness of our abstractions and the runtime performance evaluation shows negligible overhead. With a unified interface that increases potential data and coding contributors, our framework accelerates the development and deployment of larger-scale decentralized data collaboration solutions to unlock the true value of data.

1 Introduction and Problem Definition

Data collaboration is a common need in many research fields including finance, health care, cybersecurity, and many others. To protect data privacy in such collaborations, many existing protocols and systems target to avoid direct sharing of data while still achieving the required computation. However, a secure and private deployment of these new designs are often challenging. There lacks a universal framework with a consistent data collaboration workflow that reduce the difficulty and cost of such design. In this work, we identify three major types of **deployment challenges**.

- **Practicability.** Compared with protocol simulations that focus on correctness and performance evaluation, a real-world privacy-preserving system needs great engineering efforts to be practical. In most solutions, different participants need to be able to discover and identify each other through WAN. They often need to build secure and authenticated connections for communication, data storage and transmission, and performing various computations. Building such a real-world solution requires a system that manages authenticated users and their roles, database, network communication, computation runtime, and many other resources.

- **Compatibility.** The implementation of one protocol often depends on the functionality of many other protocols. For example, federated learning often requires secure aggregation as its building block to merge the gradients in a privacy-preserving way. Secure aggregation might further requires synchronization for randomness. However, different protocols are often implemented with different library dependencies in different programming languages, making it hard for integration. Even protocols implemented in the same programming language might use different data exchange formats. Poor compatibility not only slows down the integration but also makes the upgrade of the protocol difficult, it also discourages developers from replacing one protocol with a similar but better fitting one.

- **Publicity.** Knowing and understanding the terms to describe the protocol requires comprehensive background knowledge in the field of cryptography. Sometimes there are multiple implementations available with various differences that can be only discovered after testing. There lacks a unified way to describe the functionalities and security guarantees of various protocols and an index that supports searching and comparison for them.

Targeting to accelerate the deployment of decentralized data science, in this work, we build a new programming framework named CoLink that facilitates a more intuitive development and deployment workflow for decentralized solutions. Specifically, the key contribution of this work is three-fold:

1. We envision a more intuitive workflow for building decentralized solutions, defining participant roles, and specifying their responsibilities.

2. We built a programming framework named CoLink that allows protocol designer to write protocol in multiple programming languages and quickly test their ideas with easy packaging and integration with existing decentralized building blocks in different languages.

3. We evaluate the proposed framework and the case study shows significant effort reduction in protocol development with negligible performance decrease.

2 Related Work

Although there is no previous work targeting the exact same direction, there are many works focusing on relevant or orthogonal goals that provide insights for the design of CoLink. In this section, we first discuss a list of decentralized data science applications that explain the characteristics of the target workload. Then, we discuss existing work on decentralized abstractions which is relevant to the construction in CoLink.

2.1 Decentralized Data Science Apps

Federated Learning. Federated learning [16, 36, 11] is one of the most well-known decentralized data applications. It trains a machine learning model using data from different participants without exchanging the original data instances. With secure aggregation protocols [6] for merging gradients, it achieves better privacy guarantees by hiding privacy-leaking updates from the individual. There are also existing works [5, 26, 25, 14] focusing on system construction to improve communication efficiency or improving the overall learning algorithms [15]. In addition to learning logistic regression or neural-network-based models, tree-based models [8, 35, 28] are also often considered, especially when input features are separate vertically across participants.

Federated learning solution developers can potentially benefit from CoLink. Because the user and communication abstractions in CoLink are helpful in orchestrating complicated communication structures [5] between coordinators in FL servers and the clients who hold the data. With compatibility with existing secure aggregation protocol implementation, the protocol designer can focus on the machine learning part and easily add and replace secure aggregation building blocks. The final packaged CoLink-compliant protocol could be used by the developer who does not understand protocol details but still want to use it in their software products.

Federated Analytic. As mentioned in the early federated learning paper [5], federated analytics [32] that aggregate various statistics can also be an important decentralized data application with value for real-world deployment. For example, Google uses federated analytics to evaluate its word prediction model on the client's GBoard app [1]. Instead of focusing on the learning of a machine learning model, it focuses on "collaborative data science without data collection" [1] which presents an overlap with the goal in this work. Based on this observation, in CoLink, we also consider the asymmetric role assignment in the decentralized setting and also support protocols with various network topology.

Secret Sharing and Secret Management. Secret sharing [27] is a common building block for many multi-party computation-based protocols. In real-world systems, when combined with other techniques, it can be also used for storage [23]. To further protect the confidentiality of the shared secrets, Refresh of the secrets [9] is helpful for long-term protection.

Private Query Processing. In addition to storing the data, there are also existing works targeting to allow query functionalities to data that is securely stored or secret-shared. CryptDB [22] allows one party to store encrypted data on another party but still be able to perform queries on those data. Splinter [33] focuses more on query obliviousness and performs queries on secret shared data on non-colluding servers to hide query access patterns from the data owners. There are also many other solutions [21, 30, 3] with various settings but all target to support a privacy-preserving database for one or multiple data owners.

Privacy-preserving Machine Learning. In addition to the model training in federated learning, there are also existing works focusing on privacy-preserving machine learning [19] that present a more strict limitation for information being exchanged during both the inference and the training of the model. As the performance of the privacy-preserving inference becoming practical [4, 24, 18, 31, 29], its communication pattern is also considered in the design of CoLink.

2.2 Decentralized Abstractions

Many existing works also propose to help the deployment of decentralized data science and provide various abstractions. Here we discuss a few relevant solutions that are either similar to the construction in CoLink or could be used together with CoLink for additional functionalities.

Global data plane [20] provides storage and communication-related abstraction in a distributed setting but focuses more on the data provenance tracking. CALYPSO [13] uses distributed ledgers as its storage building block and focuses on monetizing auditable data access with access control. Memri [17] targets to boost private and trustworthy use of data and allows users to manage their data themselves. Gretel [10] improve privacy for accessing data using differential privacy and data synthesis and provide users with API to simplify development.

There are also works focusing on strengthening privacy and security in different decentralized applications. Data Capsule [34] allows automated privacy analysis for regulation compliance using a policy language. Viaduct [2] builds a compiler for secure distributed programs and maps an application program to the language in its MPC or ZKP backends.

3 Approach

3.1 CoLink Workflow

We build CoLink to accelerate the development of security protocols and overcome the deployment obstacles. We observe that, in most decentralized data collaborations, participants manage their own data as input, communicate with each other, and execute privacy-preserving protocols to get separate output. To be aligned with the target usage, as shown in Figure 1, we propose a new unified workflow to describe and implement collaborative protocols.

In the new workflow, each participant starts a CoLink server in their trusted environment and store their private data in it. The CoLink server acts as a data asset manager on behalf of the users who host their data on it. To start a protocol, one participant instantiate a task in the system by calling their own CoLink server endpoint and specifying who they want to collaborate with. Their CoLink server then reaches out representing them to invite other CoLink servers to join the task. Once all participants agree to proceed, the CoLink servers from all participants start the protocol instance for the task.

To support such a workflow, CoLink provides a unified interface to manage user, storage, communication, and computation. By extending gRPC, we simplify the development of multi-party protocols and orchestrate implementations in different programming languages to work together consistently. With a unified interface that increases potential data contributors, our framework has the potential to enable larger-scale decentralized data collaboration.

Two types of programmers can benefit from CoLink. Data collaboration *application developers* who understand the functionalities and threat models of security protocols can directly integrate off-the-shelf protocols as black boxes into their application system. And *protocol designers* who are often researchers or cryptography experts save efforts on building repeated basic abstractions and writing reusable protocol implementation code.

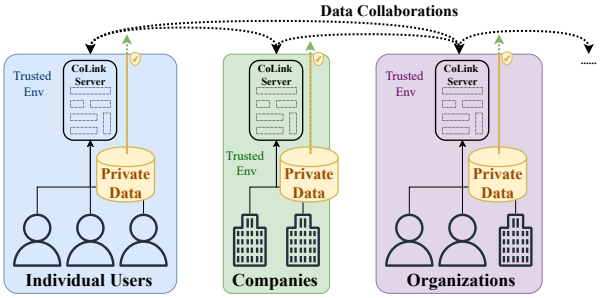

Figure 1: Unified collaboration workflow with CoLink.

3.2 CoLink Components

We design a plug-in architecture for CoLink shown in Figure 2. At a high level, each participant client has a *CoLink server* hosted on their laptop or in a trusted cloud environment and store their data there. CoLink servers manage the storage and data collaboration requests, ask the client to make decisions on whether to approve a collaboration request, and communicate with other CoLink servers to run protocols on behalf of their owner.

Each user's framework contains three components:

- **CoLink Server.** To use CoLink in the programming environment, each client needs to setup, in an environment they trust, a CoLink server that manages the user, storage, and network-related abstractions. For scalability, multiple clients may share the same CoLink server as the server provides user-based isolation. Inside the CoLink server, there is a key-value store for persistent storage, a core scheduler to handle various requests, an inter-core communication module to connect with other CoLink servers, a gRPC service, and a message queue to handle connections from both the client and other CoLink servers with another layer of access control.

- **CoLink SDKs.** To simplify the programming experience for both application developers and protocol designers, we provide SDKs in various programming languages for writing programs that can interact with the CoLink Server. Specifically, for each language, we target to provide two different SDKs: *an application SDK (SDK-A)* for application developers and *a protocol SDK (SDK-P)* for protocol designers. SDK-A allows the client to update storage, manage computation requests, and monitor

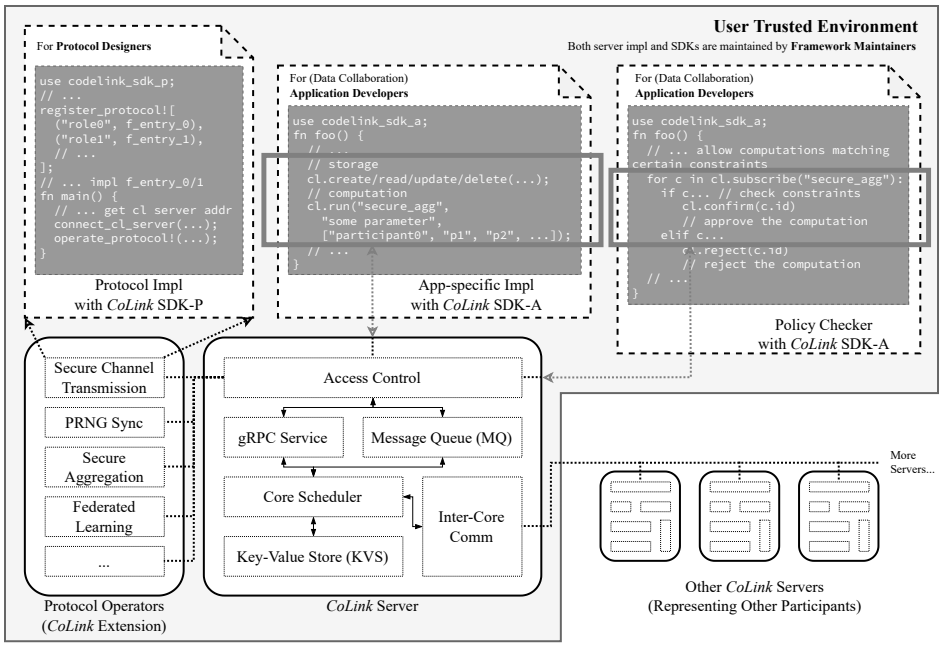

Figure 2: CoLink Architecture.

CoLink server status. And SDK-P allows protocol designers to write CoLink Extensions that extend the functionality of CoLink with new protocols.

- **Protocol Operators (CoLink Extensions).** According to our design principle of being extensible, we detach the implementation of specific protocol from the CoLink server. With SDK-P, a protocol designer can write a program that connects to the CoLink server to register itself as a protocol operator. Whenever someone requests a protocol execution from the CoLink server and gets approved, the CoLink server forwards the request to the corresponding operator.

With the above three components, CoLink provides the necessary programming abstraction to handle computation and can link code in different programming languages from different participants together. By providing SDK-P in different programming languages, CoLink can integrate protocols implemented in different languages as operators. By having SDK-A in different programming languages, the client can use whatever environment they prefer to start certain protocol executions.

3.3 CoLink Programming Abstractions

In this section, we explain the gRPC endpoints defined by CoLink server to support various abstractions. To reduce the learning efforts, we maintain a minimal set of 14 endpoints, including 3 for user and authentication, 7 for storage, 3 for tasks, and only 1 for inter-server communication. Next, we describe those endpoints in detail. Note that the CoLink SDKs expose the same interface in the target programming language.

3.3.1 Connection and User Authentication

CoLink uses JSON Web Token (JWT) for client authentication and TLS for server authentication with a mutual TLS option. JWT contains both a user ID and its role, which could be the administrator (admin), a normal user, or a guest. When the CoLink server starts, it stores a default admin JWT to a location where only the infrastructure manager could access.

1. **RequestCoreInfo**(): Serving as a liveness checking endpoint, RequestCoreInfo provides the basic background information about the current server, including the public key of the server and the URI for the message queue. Note that it is the only endpoint that does not require any JWT.

2. **ImportUser**(*UserConsent*): Decided to host data on the server, user construct a user consent by signing the public key of the server and ask the admin to submit a ImportUser request with admin JWT on behalf of them. The returned user JWT is used in the following operations.

3. **RefreshToken**(*ExpirationTime*): This endpoint allow both the admin and users to refresh their JWT token updating the field of the expiration time.

3.3.2 Storage

CoLink maintains an internal in-memory key-value storage for both user data and server status. There are two major difference between a normal key-value storage and CoLink storage. First, CoLink maintains the complete history for all storage entries, implying that the deletions are only special update operations. To reflect this difference, each storage entry has a *KeyPath* which uniquely identify an immutable value and a *KeyName* that point to the value for the latest KeyPath. Specifically, KeyPath takes the form of **"UserID::layer1:...:layerN:varname@timestamp"** and **"layer1:...:layerN:varname"** is the KeyName.

4. **CreateEntry**(*KeyName*, *Value*): It creates a new entry in CoLink storage. Specifically, it use the current timestamp and the user ID loaded from JWT to construct a KeyPath and insert the value. The endpoint returns the contructed KeyPath.

5. **ReadEntries**(*ListOfKeys*): Returns the values in a list of KeyNames or KeyPaths. For KeyName inputs, it find the latest matching KeyPath and return it along with the value inside.

6. **UpdateEntry**(*KeyName*, *Value*): Add a new KeyPath with the latest value for the KeyName. It creates a new entry if the KeyName does not exist.

7. **DeleteEntry**(*KeyName*): Instead of removing the value completely, DeleteEntry only redirects the KeyName to an empty location. Whoever knows the KeyPath will still have access to the value.

8. **ReadKeys**(*Prefix*, *IncludeHistory*): In addition to the basic CRUD operations, ReadKeys further allows user to iterate through a sub-space in the storage specified by a give prefix. The boolean flag IncludeHistory indicates whether the return is in the KeyPath or KeyName level.

9. **Subscribe**(*KeyName*, *StartTimestamp*): Subscribe allow users to monitor the state changes on the server. By specifying the storage entry to be monitored, the server returns a MQ handler that contains messages related to all changes for the given KeyName after a certain StartTimestamp, avoiding the resource consuming client-side polling.

10. **Unsubscribe**(*Handler*): If the user no longer cares about a specific subscription, Unsubscribe helps reduce the server burden of monitoring the changes and sending message for notifications.

3.3.3 Decentralized Task

CoLink provide an abstraction of decentralized task. It contains a TaskID, the protocol type, participant and their roles in the protocol, and many other metadata. The status of the tasks are maintained in the storage and can be accessed by operations in Section 3.3.2. Protocol operators get updates about incoming task execution request by subscribing to a special storage location.

11. **CreateTask**(*Task*): The initiator specifies the task details. CreateTask helps the user initiate the task and send invitations to other participants.

12. **ConfirmTask**(*TaskID*, *Decision*): After receiving task invitations and viewing their details from the storage, a user may use ConfirmTask to approve, reject, or ignore a task invitation. If the task is set to require a confirmation, it only starts after all participants have confirmed and approved the task.

13. **FinishTask**(*TaskID*, *Status*): Users can use FinishTask to mark that a task is finished after following through the complete execution of the protocol.

3.3.4 Inter-server Task Synchronization

Before inter-server synchronization, the receiver needs to generate a guest JWT using their user JWT and share it with the sender for authorization.

14. **InterCoreSyncTask**(*Task*): There is only one endpoint for inter-server task synchronization that serves multiple purposes. If the receiver does not have the task stored before, the request is regarded as a task invitation. If the receiver knew about the task, the request should contain signed decisions to update the task record kept by the receiver.

If the task is set to require a confirmation, the initiator first synchronize its task proposal with other participants. Other participants later synchronize the task back with their decisions after the owner calls ConfirmTask. Upon receiving approval decisions from all participants, the initiator sends the latest task with all signed decisions to others to announce the start of the task.

4 Evaluation

In this section, we conduct a case study to evaluate the effectiveness of our abstractions and profile the performance of different system steps and components.

4.1 Case Study

To show the simplicity of using CoLink, in Listing 5, we demonstrate how to write a simple collaboration protocol under CoLink. As we can see from the example, CoLink provide simple storage-related abstraction (line 29) using SDK-A. For more detailed usage of SDK-A, we more coding snippets in Appendix A. Protocol designer can integrate their existing work by adding entry functions for each role (line 7 and 21). Serving the function entries with the macro in SDK-P, the code can run as a protocol operator as mentioned in Section 4. With CoLink SDK-P, protocol designers can better focus on the functionality development and reduce unnecessary and repetitive efforts on basic abstractions.

Listing 1: CoLink Protocol Example in Rust

```
1  #![allow(unused_variables)]
2  use colink_sdk_a::{CoLink, Participant};
3  use colink_sdk_p::ProtocolEntry;
4
5  struct Initiator;
6  #[colink_sdk_p::async_trait]
7  impl ProtocolEntry for Initiator {
8      async fn start(
9          &self,
10         cl: CoLink,
11         param: Vec<u8>,
12         participants: Vec<Participant>,
13     ) -> Result<(), Box<dyn std::error::Error>> {
14         println!("initiator");
15         Ok(())
16     }
17 }
18
19 struct Receiver;
20 #[colink_sdk_p::async_trait]
21 impl ProtocolEntry for Receiver {
22     async fn start(
23         &self,
24         cl: CoLink,
25         param: Vec<u8>,
26         participants: Vec<Participant>,
27     ) -> Result<(), Box<dyn std::error::Error>> {
28         println!("{}", String::from_utf8_lossy(&param));
29         cl.create_entry(&format!("tasks:{}:output",
               cl.get_task_id()?), &param)
30             .await?;
31         Ok(())
32     }
33 }
34
35 colink_sdk_p::protocol_start!(
36     "greetings",            // protocol_name
37     ("initiator", Initiator), // bind initiator's entry
38     ("receiver", Receiver)  // bind receiver's entry
39 );
```

4.2 Performance Profiling

We evaluate the performance of our solution. First, we measure the CPU and memory usage when the system is idle. As shown in Table 1, all system components have very low resource consumption when not activated, which suggest a low abstraction overhead. Second, we measure the average latency of the common system operations. The user import operation takes 520ms and the storage operations only take 10ms on average. This suggests that dozens for storage operations are often negligible for data collaborations that takes minutes if not days.

	CPU Usage	VSZ	RSS
CoLink Server	0.0	5.3MB	22.5KB
Policy Module	0.0	10.6MB	26.0KB
Protocol Operator	0.0	10.6MB	41.3KB

Table 1: CPU and Memory Usage

5 Conclusion and Future Work

In this work, we introduce CoLink, a programming abstraction to simplify the deployment of decentralized data science. The case study shows the effectiveness of our abstraction in simplifying the solution system design. And the runtime performance evaluation shows the negligible performance impact of our additional abstraction.

In the future, we plan to add more built-in protocol operators for our framework to further reduce the burden of developers. For example, to further increase the publicity mentioned in Section 1, we plan to write CoLink protocol that acts as a data registry. Such a registry can be helpful in the client discovery procedure and further simplify the programming abstraction. We will also investigate the integration possibility with popular existing MPC (e.g. [12]) or ZKP (e.g. [7]) tool-kits in the future.

References

[1] Federated analytics: Collaborative data science without data collection. `https://ai.googleblog.com/2020/05/federated-analytics-collaborative-data.html`. Accessed: 2020-05-27.

[2] ACAY, C., RECTO, R., GANCHER, J., MYERS, A. C., AND SHI, E. Viaduct: an extensible, optimizing compiler for secure distributed programs. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation* (2021), pp. 740–755.

[3] BATER, J., ELLIOTT, G., EGGEN, C., GOEL, S., KHO, A. N., AND ROGERS, J. Smcql: Secure query processing for private data networks. *Proc. VLDB Endow. 10*, 6 (2017), 673–684.

[4] BOEMER, F., COSTACHE, A., CAMMAROTA, R., AND WIERZYNSKI, C. ngraph-he2: A high-throughput framework for neural network inference on encrypted data. In *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography* (2019), pp. 45–56.

[5] BONAWITZ, K., EICHNER, H., GRIESKAMP, W., HUBA, D., INGERMAN, A., IVANOV, V., KIDDON, C., KONEČNÝ, J., MAZZOCCHI, S., MCMAHAN, B., ET AL. Towards federated learning at scale: System design. *Proceedings of Machine Learning and Systems 1* (2019), 374–388.

[6] BONAWITZ, K., IVANOV, V., KREUTER, B., MARCEDONE, A., MCMAHAN, H. B., PATEL, S., RAMAGE, D., SEGAL, A., AND SETH, K. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC*

Conference on Computer and Communications Security (2017), pp. 1175–1191.

[7] BÜNZ, B., BOOTLE, J., BONEH, D., POELSTRA, A., WUILLE, P., AND MAXWELL, G. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 315–334.

[8] CHENG, K., FAN, T., JIN, Y., LIU, Y., CHEN, T., PAPADOPOULOS, D., AND YANG, Q. Secureboost: A lossless federated learning framework. *IEEE Intelligent Systems 36*, 6 (2021), 87–98.

[9] DESMEDT, Y., AND JAJODIA, S. Redistributing secret shares to new access structures and its applications. Tech. rep., Citeseer, 1997.

[10] GRETEL AUTHORS. gretel.ai.

[11] KAIROUZ, P., MCMAHAN, H. B., AVENT, B., BELLET, A., BENNIS, M., BHAGOJI, A. N., BONAWITZ, K., CHARLES, Z., CORMODE, G., CUMMINGS, R., ET AL. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning 14*, 1–2 (2021), 1–210.

[12] KELLER, M. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (2020).

[13] KOKORIS-KOGIAS, E., ALP, E. C., GASSER, L., JOVANOVIC, P., SYTA, E., AND FORD, B. Calypso: Private data management for decentralized ledgers. *Cryptology ePrint Archive* (2018).

[14] KONEČNÝ, J., MCMAHAN, H. B., YU, F. X., RICHTÁRIK, P., SURESH, A. T., AND BACON, D. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492* (2016).

[15] LI, T., SAHU, A. K., ZAHEER, M., SANJABI, M., TALWALKAR, A., AND SMITH, V. Federated optimization in heterogeneous networks. *Proceedings of Machine Learning and Systems 2* (2020), 429–450.

[16] MCMAHAN, B., MOORE, E., RAMAGE, D., HAMPSON, S., AND Y ARCAS, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics* (2017), PMLR, pp. 1273–1282.

[17] MEMRI AUTHORS. memri.io.

[18] MISHRA, P., LEHMKUHL, R., SRINIVASAN, A., ZHENG, W., AND POPA, R. A. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)* (2020), pp. 2505–2522.

[19] MOHASSEL, P., AND ZHANG, Y. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)* (2017), IEEE, pp. 19–38.

[20] MOR, N. *Global data plane: A widely distributed storage and communication infrastructure*. University of California, Berkeley, 2019.

[21] PAPPAS, V., KRELL, F., VO, B., KOLESNIKOV, V., MALKIN, T., CHOI, S. G., GEORGE, W., KEROMYTIS, A., AND BELLOVIN, S. Blind seer: A scalable private dbms. In *2014 IEEE Symposium on Security and Privacy* (2014), IEEE, pp. 359–374.

[22] POPA, R. A., REDFIELD, C. M., ZELDOVICH, N., AND BALAKRISHNAN, H. Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), pp. 85–100.

[23] RAMAN, R. K., AND VARSHNEY, L. R. Distributed storage meets secret sharing on the blockchain. In *2018 Information Theory and Applications Workshop (ITA)* (2018), IEEE, pp. 1–6.

[24] RATHEE, D., RATHEE, M., KUMAR, N., CHANDRAN, N., GUPTA, D., RASTOGI, A., AND SHARMA, R. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (2020), pp. 325–342.

[25] ROMANINI, D., HALL, A. J., PAPADOPOULOS, P., TITCOMBE, T., ISMAIL, A., CEBERE, T., SANDMANN, R., ROEHM, R., AND HOEH, M. A. Pyvertical: A vertical federated learning framework for multi-headed splitnn. *arXiv preprint arXiv:2104.00489* (2021).

[26] RYFFEL, T., TRASK, A., DAHL, M., WAGNER, B., MANCUSO, J., RUECKERT, D., AND PASSERAT-PALMBACH, J. A generic framework for privacy preserving deep learning. *arXiv preprint arXiv:1811.04017* (2018).

[27] SHAMIR, A. How to share a secret. *Communications of the ACM 22*, 11 (1979), 612–613.

[28] TIAN, Z., ZHANG, R., HOU, X., LIU, J., AND REN, K. Federboost: Private federated learning for gbdt. *arXiv preprint arXiv:2011.02796* (2020).

[29] TRAMER, F., AND BONEH, D. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287* (2018).

[30] VOLGUSHEV, N., SCHWARZKOPF, M., GETCHELL, B., VARIA, M., LAPETS, A., AND BESTAVROS, A. Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019* (2019), pp. 1–18.

[31] WAGH, S., TOPLE, S., BENHAMOUDA, F., KUSHILEVITZ, E., MITTAL, P., AND RABIN, T. F: Honest-majority maliciously secure framework for private deep learning. *Proceedings on Privacy Enhancing Technologies 2021*, 1 (2021), 188–208.

[32] WANG, D., SHI, S., ZHU, Y., AND HAN, Z. Federated analytics: Opportunities and challenges. *IEEE Network* (2021).

[33] WANG, F., YUN, C., GOLDWASSER, S., VAIKUNTANATHAN, V., AND ZAHARIA, M. Splinter: Practical private queries on public data. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)* (2017), pp. 299–313.

[34] WANG, L., NEAR, J. P., SOMANI, N., GAO, P., LOW, A., DAO, D., AND SONG, D. Data capsule: A new paradigm for automatic compliance with data privacy regulations. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*. Springer, 2019, pp. 3–23.

[35] WU, Y., CAI, S., XIAO, X., CHEN, G., AND OOI, B. C. Privacy preserving vertical federated learning for tree-based models. *arXiv preprint arXiv:2008.06170* (2020).

[36] YANG, Q., LIU, Y., CHEN, T., AND TONG, Y. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST) 10*, 2 (2019), 1–19.

Appendix

A SDK-A Coding Snippets

Here are some examples of using Rust SDK-A to interact with the CoLink server by accessing the endpoint specified in Section 3.3. The programmer connects to a CoLink server by creating a new CoLink instance (Listing 2). With the created CoLink instance, one can perform storage CRUD operations (Listing 3). We utilize the `import_user` for user initiation. We first generate a public/secret key pair, sign the expiration timestamp and the server public key, and call the `import_user` endpoint (Listing 4). Once we have created or imported the participants of a protocol, we can pass in the name of the protocol, the message, the participants of the protocol to run it (Listing 5).

Listing 2: Connect to CoLink Service

```
let cl = CoLink::new(addr, jwt);
```

Listing 3: Access Storage

```
// CRUD operations
cl.create_entry(&keyname, &value).await?;
// Read entries are done in batch.
let results = cl.read_entries(&keynames).await?;
cl.update_entry(&keyname, &value).await?;
cl.delete_entry(&keyname).await?;
```

Listing 4: Initializing User

```
// Generate a public/secret key pair.
let (pk, sk) = generate_user();
let (_, core_pub_key) = cl.request_core_info().await?;
let (signature_timestamp, sig) =
     prepare_import_user_signature(&pk, &sk, &core_pub_key,
     expiration_timestamp);
let user_jwt = cl.import_user(&pk,signature_timestamp,
     expiration_timestamp, &sig).await?
```

Listing 5: Run Multi-party Protocols

```
// Decclare Participants
let participants = vec![
    Participant {
        user_id: user_id_a.to_string(),
        ptype: "initiator".to_string(),
    },
    Participant {
        user_id: user_id_b.to_string(),
        ptype: "receiver".to_string(),
    },
];
let cl = CoLink::new(addr, jwt_a);
// Run protocol
let task_id = cl
    .run_task("greetings", msg.as_bytes(), &participants,
         true).await?;
```